UNIVERSIDAD DE LA SABANA

Presentado por:

Yuly Dayana Rodriguez Salcedo	305314
Andres Camilo Torres Guerrero	322926
Juan Felipe Herrera Pinzón	319079
Juan David Borda Munevar	322266

Presentado a:

Gabriel Hernando Soche Umaña

Informe de proyecto – Resolución de laberintos

Estructuras de datos y algoritmos, 2024-2

20 Noviembre 2024

OBJETIVOS

Objetivo General

Desarrollar un programa en Python que permita resolver laberintos de forma funcional y gráfica, aplicando estructuras de datos y algoritmos aprendidos durante el semestre, optimizando su desempeño mediante técnicas de backtracking y análisis de la complejidad computacional.

Objetivos Específicos

♦ Implementar estructuras de datos como pilas, colas y nodos, adaptándolas a los requerimientos del proyecto.

♦ Integrar técnicas de backtracking para encontrar y optimizar las soluciones al problema del laberinto.

♦ Analizar la complejidad de los algoritmos implementados utilizando la notación Big O, evaluando el desempeño del programa.

♦ Emplear la librería .time para medir y registrar los tiempos de ejecución, con el fin de evaluar la eficiencia de la solución.

♦ Diseñar una interfaz gráfica funcional que permita visualizar la solución del laberinto de manera clara y dinámica.

♦ Consolidar conocimientos teóricos y prácticos en programación, estructuras de datos y resolución de problemas mediante la ejecución del proyecto en equipo.

♦ Promover el buen uso de Python, implementando buenas prácticas de programación y aprovechando las capacidades del lenguaje para garantizar un código limpio y eficiente.

♦ Mejorar la lógica y las habilidades de programación para abordar de manera eficaz la resolución de problemas complejos.

DESCRIPCIÓN DE LA SOLUCIÓN

Conceptos a utilizar:

1. Pilas: Una estructura de datos tipo LIFO (Last In, First Out), donde el último elemento agregado es el primero en ser removido, útil para resolver problemas como el balanceo de paréntesis o el recorrido de árboles.

2. Colas: Una estructura de datos tipo FIFO (First In, First Out), donde el primer elemento agregado es el primero en ser removido, ideal para gestionar tareas en sistemas de procesamiento por lotes o simulación de procesos.

3. Nodos: Son elementos básicos de estructuras como listas enlazadas, árboles y grafos, que contienen datos y referencias (enlaces) a otros nodos, permitiendo organizar y manipular colecciones de datos de manera dinámica.

4. Backtracking: Un algoritmo de búsqueda que explora todas las soluciones posibles de manera recursiva, retrocediendo (backtracking) cuando se detecta que una solución parcial no es válida, aplicable en problemas como el Sudoku o la resolución de laberintos.

5. La notación Big O: e
Es una herramienta matemática que se usa en ciencias computacionales para analizar y describir el rendimiento de un algoritmo. Se trata de una simbología que permite medir el tiempo de ejecución, el uso de recursos y el espacio en disco de un algoritmo.

Descripción y Flujo de la solución:

El código resuelve laberintos representados como cuadrículas mediante backtracking, utilizando una pila para explorar caminos desde una posición inicial (0) hasta una posición final (X). Las paredes se representan con + y las celdas transitables con espacios. Durante la ejecución, el laberinto se visualiza en consola, destacando la posición actual con @ y el camino recorrido con o. El programa permite al usuario definir el tamaño del laberinto, la representación del mismo y un retraso en milisegundos para animar el progreso. Todas las soluciones posibles se encuentran y se almacenan, mientras se calculan estadísticas como el tiempo para encontrar cada solución, la longitud del camino y el promedio. Finalmente, se destacan la solución más corta y la más larga con su visualización correspondiente. El diseño garantiza que cada celda sea visitada una sola vez por camino y que no se queden caminos sin explorar.

Estructuras de datos utilizadas y sus ventajas:

El programa utiliza varias estructuras de datos clave. La **pila (Stack)** implementa el backtracking, permitiendo explorar caminos de forma eficiente y retroceder cuando no hay movimientos válidos. Un **conjunto (visited)** asegura que no se formen ciclos al almacenar las celdas ya visitadas. Aunque no se usa directamente en esta implementación, un **árbol (TreeNode)** es útil para representar múltiples caminos jerárquicamente, con cada nodo almacenando coordenadas, su padre y sus hijos. Las soluciones se encapsulan en objetos que contienen el camino, el tiempo tomado y la longitud. La matriz (maze) facilita el acceso y la modificación del laberinto en tiempo real durante la exploración. Estas estructuras, junto con su interacción, permiten una solución modular y escalable, ideal para personalización o para incorporar métodos alternativos como búsqueda en anchura (BFS) para optimizar el proceso.

ANÁLISIS DE COMPLEJIDAD Y ORDEN (Notación Big O)

1. **Inicialización y Verificación del Laberinto**:
    - **Complejidad**: O(n^2)
    - **Explicación**: La matriz del laberinto se inicializa a partir de una cadena de longitud n^2 (donde n es el tamaño del laberinto) y se verifican las posiciones inicial y final.

2. **Búsqueda en Profundidad (DFS) utilizando Pila**:
    - **Complejidad del peor caso**: O(V + E)
        - V: Número de vértices (celdas en el laberinto, máximo n^2).
        - E: Número de aristas (movimientos posibles entre celdas).
    - **Explicación**: El DFS explora cada celda al menos una vez. Si el laberinto está muy conectado (con muchas celdas transitables), cada movimiento se realiza al verificar las celdas vecinas.

3. **Verificación de Vecinos Válidos**:
    - **Complejidad**: O(1) para cada celda.
    - **Explicación**: La verificación de si una celda es válida se realiza comprobando solo cuatro posibles direcciones (arriba, abajo, izquierda, derecha).

4. **Impresión del Laberinto**:
    - **Complejidad**: O(n^2) cada vez que se imprime.
    - **Explicación**: Cada impresión del laberinto involucra recorrer toda la matriz n x n.

5. **Construcción de Soluciones**:
    - **Complejidad**: O(k) para cada camino encontrado (donde k es la longitud del camino).
    - **Explicación**: Cada camino se almacena en una lista, lo cual depende directamente del número de celdas en el camino.

6. **Complejidad General del Programa:**
    - **Complejidad:** Cuando el DFS con impresión frecuent**e** domina el costo total en el peor caso, el resultado es una complejidad global de O(n^4)
    - **Explicación:** Sin impresión constante, el DFS y las operaciones asociadas tendrán un grado de complejidad de O(n^2**)** por tanto, el laberinto es polinómico en tiempo y su grado más elevado está relacionado con el costo de impresión o almacenamiento en escenarios extremos.

DICCIONARIO DE VARIABLES Y MÉTODOS

NOMBRE	TIPO	DESCRIPCIÓN
size	Entero	Tamaño del laberinto (n x n).
maze_string	Cadena	Representación del laberinto como una cadena de n^2 caracteres.
delay	Entero	Retraso en milisegundos para la animación de pasos.
maze	Lista de Listas	Representación 2D del laberinto a partir de maze_string.
start	Tupla	Coordenadas iniciales del laberinto (donde comienza la búsqueda).
end	Tupla	Coordenadas finales del laberinto (objetivo de la búsqueda).
visited	Conjunto	Conjunto de celdas ya visitadas.
solutions	Lista	Lista de soluciones encontradas en el laberinto.
current_position	Tupla	Coordenadas actuales durante la búsqueda.
_find_position	Método	Encuentra las coordenadas de un carácter específico en el laberinto.
_is_valid_move	Método	Verifica si un movimiento a una celda es válido.
_move_to	Método	Mueve el "cursor" a una celda y actualiza la visualización del laberinto.

Universidad de La Sabana

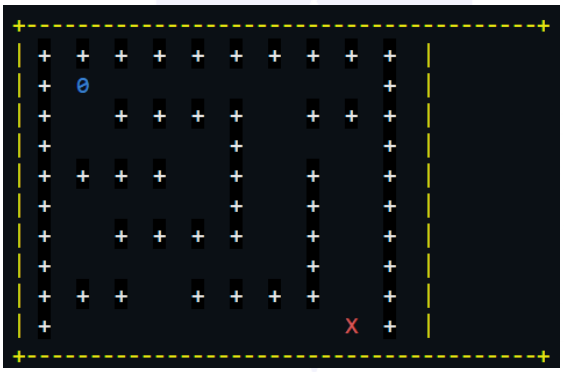
_print_maze	Método	Imprime el laberinto en la consola.
verify maze	Método	Verifica si el laberinto es válido (si tiene un inicio y fin definidos).
_ find_all solutions	Método	Realiza la búsqueda en profundidad y almacena las soluciones encontradas.
_print_final_statistics	Método	Imprime estadísticas de las soluciones encontradas.
_print_solution	Método	Imprime un camino de solución en el laberinto.
Node	Clase	Representa un nodo con coordenadas (x, y) para las estructuras de pila y cola.
Stack	Clase	Implementa una pila LIFO para almacenar las celdas durante la búsqueda.
Queue	Clase	Implementa una cola FIFO (no utilizada en este caso, pero lista para BFS).
TreeNode	Clase	Representa un nodo de un árbol (no utilizado en la solución actual).
Solution	Clase	Almacena un camino (path), el tiempo de búsqueda (time_found) y la longitud (length) de la solución.

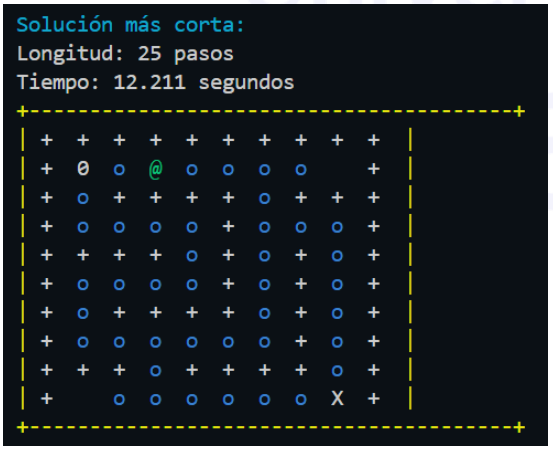
Universidad de La Sabana

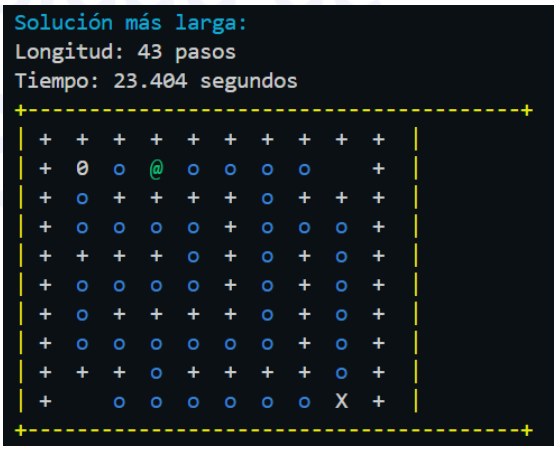
MazeSolver	Clase	Clase principal que gestiona la lógica del laberinto y la búsqueda de caminos.

EJEMPLO LABERINTO 10 x 10 DOCUMENTADO

++++++++++0 ++ ++++ ++++ + +++++ + + ++ + + ++ ++++ + ++ + ++++ ++++ ++ X+







CONCLUSIONES

El proyecto alcanzó con éxito los objetivos planteados al integrar y adaptar diversas estructuras de datos como pilas, colas y nodos, fundamentales para implementar la resolución del laberinto utilizando técnicas de backtracking, lo que permitió explorar rutas de manera eficiente y encontrar todas las soluciones posibles.

Además, se realizó un análisis detallado de la complejidad algorítmica mediante la notación Big O, lo cual brindó una mejor comprensión sobre el comportamiento del algoritmo frente a distintos tamaños de entrada, mientras que la librería **.time** permitió medir el tiempo de ejecución de manera precisa, aportando datos valiosos sobre el rendimiento del programa. La visualización gráfica del proceso de resolución contribuyó no solo a mejorar la experiencia del usuario, sino también a hacer más tangible el funcionamiento interno del algoritmo, facilitando su comprensión.

Este enfoque práctico no sólo consolidó los conocimientos adquiridos en el curso, sino que también permitió a los miembros del equipo fortalecer sus habilidades de programación, optimización de código y resolución de problemas. El cumplimiento de los requisitos establecidos en la rúbrica del proyecto evidenció la capacidad del equipo para diseñar e implementar soluciones eficientes, y el desarrollo de este solucionador de laberintos proporcionó una experiencia enriquecedora en términos de trabajo colaborativo y diseño algorítmico, preparándose para enfrentar futuros desafíos de programación

Universidad de La Sabana

REFERENCIAS

Python Software Foundation, "Python Tutorial: Data Structures," disponible en: https://docs.python.org/3/tutorial/datastructures.html

Geeks for Geeks, "Python Data Structures," disponible en: https://www.geeksforgeeks.org/python-data-structures/

Anthropic, "Claude: AI Language Model," disponible en: https://www.anthropic.com/claude